EXHIBIT 35

A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services

By Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger

Abstract

Datacenter workloads demand high computational capabilities, flexibility, power efficiency, and low cost. It is challenging to improve all of these factors simultaneously. To advance datacenter capabilities beyond what commodity server designs can provide, we designed and built a composable, reconfigurable hardware fabric based on field programmable gate arrays (FPGA). Each server in the fabric contains one FPGA, and all FPGAs within a 48-server rack are interconnected over a low-latency, high-bandwidth network.

We describe a medium-scale deployment of this fabric on a bed of 1632 servers, and measure its effectiveness in accelerating the ranking component of the Bing web search engine. We describe the requirements and architecture of the system, detail the critical engineering challenges and solutions needed to make the system robust in the presence of failures, and measure the performance, power, and resilience of the system. Under high load, the large-scale reconfigurable fabric improves the ranking throughput of each server by 95% at a desirable latency distribution or reduces tail latency by 29% at a fixed throughput. In other words, the reconfigurable fabric enables the same throughput using only half the number of servers.

1. INTRODUCTION

Cloud computing has emerged as a dominant paradigm for delivering scalable, reliable, and cost-effective online services to businesses and clients across the world. According to the IDC, public spending in IT cloud services will grow to more than $127B in 2016 as the adoption of cloud computing accelerates worldwide.[10] This major shift will offer enormous potential in unlocking new applications and in improving the performance, security, and cost of computing.

The majority of today's cloud services are realized using datacenters, which typically comprise tens if not hundreds of thousands of servers built from commodity components such as general-purpose processors, memory, storage, and networking. Datacenters are shared across applications and services, providing economies of scale, reliability, scalability, and shared infrastructure management.

Datacenter operators have traditionally relied upon continuous improvements in the performance and efficiency of

general-purpose processors to make datacenters even more powerful and cost-effective. These improvements have been largely driven by Moore's law that predicts an exponential growth in the number of transistors over time, and by Dennard scaling[8] that predicts constant power consumption even as the number of transistors increase within a fixed silicon area.

In recent years, Dennard scaling has virtually ended, resulting in power consumption being roughly proportional to the number of switching transistors. Thus, even though Moore's law continues to provide more transistors for the time being, a larger number of transistors must switch proportionally less frequently to maintain constant power consumption. In addition, as transistors become smaller, they are becoming more expensive to manufacture, making it even less attractive to pack more and more transistors onto a single chip.

1.1. Specialized hardware in the datacenter

One way to improve the performance and efficiency of datacenter servers is to make better uses of "power-limited" transistors by specializing servers and their components to a particular task. In the academic literature, specialization has been shown to achieve $10\times$–$100\times$ or more improvement in energy efficiency over general-purpose processors in many cases, such as for Memcached[5, 14] compression/decompression,[13, 15] K-means clustering,[9, 12] and parts of web search.[20] However, while specializing servers for specific workloads can provide significant efficiency gains, doing so is problematic in the datacenter for several major reasons.

First, datacenters, by their very nature, support a wide variety of applications and specializing for one service will likely cause inefficiencies and add cost to any other services sharing the platform. Second, specialization compromises homogeneity, which is highly desirable in the datacenter environment to reduce management issues and to provide a consistent platform on which applications can rely upon. Third, datacenter services evolve rapidly, making highly specialized hardware features impractical and quickly obsoleted. Thus, datacenter providers face a conundrum: they need continued improvements in performance and

The original version of this paper was published in the *Proceedings for the 41st ACM/IEEE International Symposium on Computer Architecture* (June 14–18, 2014, Minneapolis, MN), 13–24.

efficiency, but cannot obtain those improvements from any combination of standard general-purpose processors and static specialized hardware.

1.2. Flexible specialized hardware

Programmable hardware, in the form of field programmable gate arrays (FPGAs), are devices that could potentially reconcile the need for both flexibility and energy efficiency. FPGAs can be imagined as silicon "Legos"—collections of simple logic blocks that can be configured and composed to implement arbitrary circuits that run at very high efficiency relative to general-purpose processors. While less energy efficient than hardwired Application Specific Integrated Circuits (ASICs), FPGAs can be rapidly reconfigured and adapted to changing workloads over the lifetime of the server. However, as of this writing, FPGAs have not been widely deployed as compute accelerators in either datacenter infrastructure or client devices.

One challenge traditionally associated with FPGAs is the need to fit the accelerated function into the available reconfigurable area on one chip. Today's FPGAs can be virtualized using runtime reconfiguration to support more functions than could fit on a single device. However, the amount of state that needs to be saved and restored, along with current reconfiguration times for standard FPGAs make this approach too slow to be practical. Multiple FPGAs could provide more silicon resources, but are extremely difficult to fit into a conventional datacenter server. Even if sufficient space were available, multiple FPGAs per server would cost more, consume more power, are wasteful when there are more FPGAs than needed—and even less useful when there are still not enough FPGAs to implement the application. On the other hand, being restricted to a single FPGA per server restricts the workloads that might be accelerated and could make the associated gains too small to justify the cost.

1.3. Catapult reconfigurable fabric

This article describes a reconfigurable fabric called Catapult, which uses FPGAs to provide the performance and efficiency gains of specialized hardware while simultaneously satisfying the strict requirements of the datacenter. As illustrated in Figure 1, the Catapult fabric is embedded into racks of servers in the form of a small board with a medium-sized FPGA and local DRAM attached to each server. A unique characteristic of Catapult is that FPGAs are directly wired together in a high-bandwidth, low-latency network, allowing services to allocate groups of FPGAs to provide the necessary reconfigurable area to implement the desired functionality.

While proving functionality on a small number of servers shows the potential of the design, datacenters are characterized by massive scales and severe power, cost, and reliability constraints. To demonstrate the potential of this technology at datacenter scale, we tested the Catapult reconfigurable fabric, running a widely deployed web search workload augmented with failure handling, on a bed of 1632 servers equipped with FPGAs. The experiments show that large gains in search throughput and latency are achievable on a real, complex commercial workload using the large-scale reconfigurable fabric.

Compared to a pure software implementation, the Catapult fabric achieves a 95% improvement in throughput at each ranking server with an equivalent latency distribution. At the same throughput as software, Catapult reduces tail latency by 29%. In other words, adding the FPGA enables the same throughput using only half the number of servers. The system is able to run stably for long periods, with a failure handling service quickly reconfiguring the fabric upon errors or machine failures. The rest of this article describes the Catapult architecture and our measurements in more detail.

2. BACKGROUND

FPGAs are digital chips that can be programmed (and reprogrammed) for implementing complex digital logic. Conceptually, FPGAs consist of an array of programmable logic elements, connected by a programmable routing network that carries signals from where they are generated to where they are consumed. Each of these features are controlled by memory cells that are configured by the end user. Designs for an FPGA

Figure 1. Each server in a rack has a local FPGA attached to the host via PCI Express. FPGAs communicate with their neighbors using a private low-latency, high-bandwidth serial network. Multiple FPGAs can be allocated to a single service, such as the Bing ranking, without going through host CPUs.
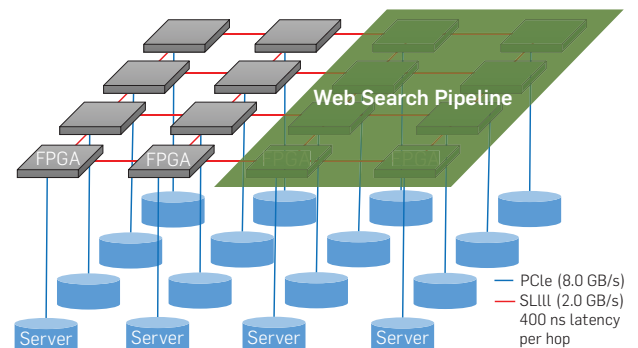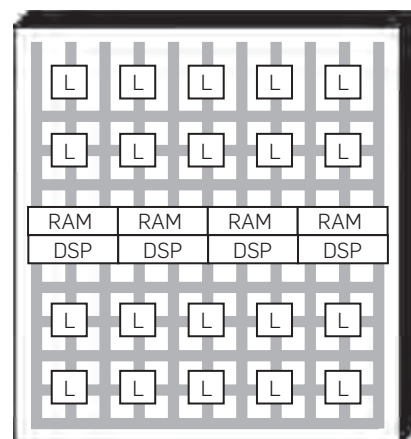

Figure 2. Block diagram representing one tile of an abstract FPGA. Each tile is composed of generic logic gates (L), embedded memory (RAMs), and specialized arithmetic units (DSPs). These basic tiles are replicated to create larger and larger FP-GAs.

are typically developed in a hardware description language (HDL) such as VHDL or Verilog; these designs are compiled down to an FPGA program (called a bitstream), similar to how a software language is compiled into a software executable. However, specifying a design in HDL is significantly more complex than for a typical software language, and compilation may take hours to complete. Once the bitstream is generated, it can be loaded into an FPGA in a matter of seconds, configuring the FPGA to implement the desired computation.

FPGAs cover a middle-ground in performance/efficiency and flexibility compared to fully custom ASICs and general-purpose CPUs. General-purpose CPUs are the most flexible platform, capable of implementing any application. But this flexibility generally comes at the cost of a 100× or greater reduction in performance and energy efficiency compared to an ASIC designed for the same task. However, when the application changes, the CPU can still run the new application by simply recompiling the software. When the target application for an ASIC changes, it likely requires designing a new chip—a process which typically takes months and carries huge development costs.

FPGAs combine aspects of both CPUs and ASICs. They can be reprogrammed as applications change to provide CPU-like flexibility, but the program produces hardware which is specialized to the application, providing performance and power efficiency closer to ASICs. The generic FPGA logic can be configured to exploit huge amounts of fine-grained parallelism and can implement very complex pipelined structures that can be several orders-of-magnitude faster and lower power than their software equivalents.

Modern FPGAs provide millions of gates of random logic and can include multiple megabytes of internal storage that supports thousands of reads and writes on every clock cycle. These chips also incorporate smaller fully custom blocks, such as complex embedded arithmetic elements (DSP blocks), high-speed I/O, IP protocol blocks, and may include complete microprocessors as well, either as hardened subsystems or by mapping the microprocessor logic into the programmable logic of the chip.

FPGAs have been available for several decades and they have huge potential for accelerating a wide variety of applications. Yet despite this promise, FPGAs have not been widely deployed as compute accelerators, even in datacenter environments where their potential for flexibility, power efficiency, and performance should make them extremely attractive.

The Catapult architecture described here unlocks the potential of FPGAs in the datacenter. To achieve this, the architecture has to be distributed, scalable, robust, and work on real-world datacenter-scale workloads. In the following sections, we describe the architecture, and show how production workloads can finally unlock the potential efficiency and performance gains promised for so long by FPGAs.

3. CATAPULT ARCHITECTURE

Datacenters are a challenging environment for any new technology to succeed. The scale alone demands extremely high reliability, efficiency, and low cost. The rapid pace of application development requires flexibility and robustness. And the physical constraints and uptime requirements make it largely impractical to modify or upgrade the hardware after initial delivery.

To succeed in the datacenter environment, an FPGA-based reconfigurable fabric must (at a minimum) meet the following requirements:

- preserve server homogeneity to avoid complex management of heterogeneous servers,
- scale to large workloads that might not fit into a single FPGA,
- avoid consuming too much power or network bandwidth,
- avoid single points of failure, and have minimal impact on reliability,
- provide positive return on investment (ROI), and
- operate within the space and power confines of existing servers.

These requirements guided the architectural choices we made throughout the Catapult system development.

3.1. Integration

How to integrate FPGAs into the datacenter is perhaps the most important consideration when designing a reconfigurable fabric. We investigated a variety of approaches which can be roughly broken into two categories based on how they integrate with conventional servers: "networked" and "integrated." Networked designs add FPGAs to special FPGA-enabled servers, and arrange the servers either as entire racks of specialized servers or embedding some number of specialized servers in otherwise conventional racks. Integrated designs add FPGAs directly inside the conventional servers, requiring no specialized servers and no network communication to reach the FPGA.

Networked designs have been developed and deployed in High Performance Computing (HPC) environments. While HPC systems are not subject to the same scale, cost, power, and homogeneity constraints as the datacenter, they are one place where integrating FPGAs with CPUs has seen some success. Entire large systems have been built using only specialized servers, including the Cray XD-1,[7] Novo-G,[11] and QP.[18] Examples of specialized servers that could be integrated with racks of conventional datacenter servers include the Convey HC-2,[6] Maxeler MPC series,[17] BeeCube BEE4,[4] and SRC MAPstation.[19] In fact, embedding a few specialized servers into each rack is the approach that we first took early in the project. However, as we learned from our first prototype, the networked design approach is inappropriate for datacenter use for several reasons.

First, specialized racks and servers are single points of failure, where the failure of the specialized nodes impacts many dependent conventional servers—amplifying the impact to uptime and overall service reliability.

Second, specialized racks and servers require separate cooling and power provisioning, as well as different software, firmware, and spare parts provisioning, making management and maintenance more difficult.

Third, all communication with these racks/servers goes through the existing network, which (in a datacenter) is not typically designed for the many-to-one communication patterns to the specialized racks.

Finally, all communication between the portion of the application running on the conventional server and the portion in the FPGA requires going over the network, which, even in the datacenter, can have high latency, unreliable delivery, and variable performance. This is particularly difficult when traffic is exhibiting the many-to-one communication pattern. The unreliable performance makes partitioning the application between CPU and FPGA extremely difficult, especially for latency-sensitive user-facing applications, and reduces the likelihood that an application can be beneficially offloaded to the reconfigurable fabric.

These issues diminish in severity with an increasingly distributed design. At the extreme is the integrated design—adding an FPGA to every server. In integrated designs, an FPGA failure only impacts one server. All servers have the same cooling and power constraints. CPU to FPGA communication does not need to go over the network at all. And attaching the FPGA directly to the CPU greatly diminishes communication latency and variance, enabling finer-grain and more reliable offloading from the CPU to the FPGA.

The homogeneous nature of the integrated design enables additional benefits which are key to keeping the designs cost-effective. Homogeneous computing resources can be divided at arbitrary granularities, easing both short-term and long-term provisioning of servers. Homogeneous designs also improve economies of scale leading to more cost-effective designs.
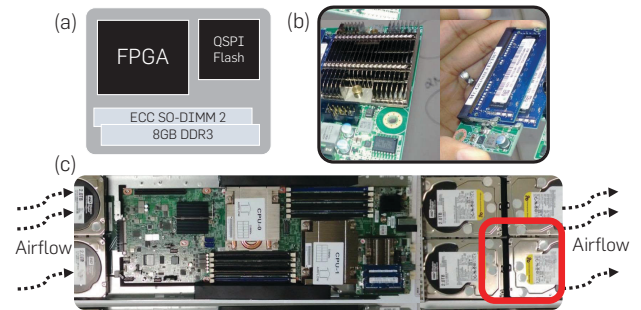
3.2. Scalability
On its own, integrating only one FPGA per server either limits applications to those that can fit into the resources of a single FPGA or suffers the same high-latency and unreliability problems with communicating over the network as specialized racks and servers.

To overcome this shortcoming, we built a specialized network, in addition to the existing Ethernet network, to facilitate FPGA-to-FPGA communication. This specialized network takes advantage of the low-latency, high-speed serial I/O available in the FPGAs to create a two-dimensional 6 × 8 torus network. The torus topology balances routability, resilience, and cabling complexity. Each inter-FPGA network link supports 20 Gbits per second in both directions, at sub-microsecond latency per hop, and requires only passive copper cables with no additional networking costs such as network interface cards or switches. Another HPC system, Maxwell,[3] takes a similar approach, but again targets HPC workloads and constraints.

3.3. FPGA board design
Figure 3 shows the FPGA board and the server into which it installs.[16] The board incorporates a midrange Altera Stratix V D5 FPGA,[2] which we selected to balance the cost of deploying thousands of FPGAs against the FPGA capacity. The board also includes two banks of DDR3-1600 DRAM. The PCIe and

Figure 3. The FPGA board and the server into which it installs. (a) Block diagram of the FPGA board. (b) Picture of the manufactured board. (c) Diagram of the 1U, half-width server that hosts the FPGA board. The air flows from left to right, leaving the FPGA in the exhaust of both CPUs.

inter-FPGA network are wired to a connector on the bottom of the board that plugs directly into the motherboard. To avoid changes to the server itself, we custom designed the board to fit within a small 10 cm × 9 cm × 16 mm slot occupying the rear of a 1U (4.45 cm high), half-width server, which offered sufficient power and cooling only for a standard 25-Watt PCIe peripheral device. These physical constraints are challenging for FPGAs, but are prohibitive for modern compute-class Graphics Processing Units (GPUs).
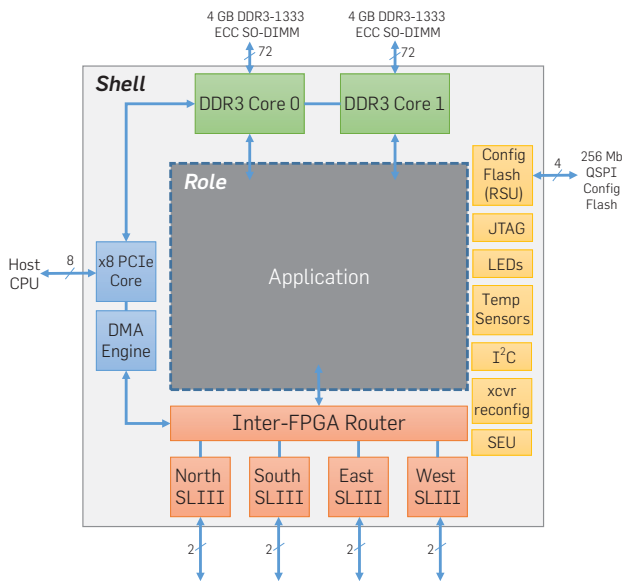
3.4. Shell architecture
In typical FPGA programming environments, the user is often responsible for developing not only the application itself but also building and integrating system functions required for data marshaling, host-to-FPGA communication, and inter-chip FPGA communication (if available). System integration places a significant burden on the user and can often exceed the effort needed to develop the application itself. This development effort is often not portable to other boards, making it difficult for applications to work on future platforms.

Motivated by the need for user productivity and design re-usability when targeting the Catapult fabric, we logically divide all programmable logic into two partitions: the *shell* and the *role*. The *shell* is a reusable portion of programmable logic common across applications targeting the same board—while the *role* is the application logic itself, restricted to a large fixed region of the chip.

Figure 4 shows a block-level diagram of the shell architecture, consisting of PCIe with a custom DMA engine, two DRAM controllers, four high-speed inter-FPGA links running the Serial Lite III protocol, the torus network router, reconfiguration logic, single event upset (SEU) scrubbing, and debugging interfaces.

Role designers access convenient and well-defined interfaces and capabilities in the shell (e.g., PCIe, DRAM, routing, etc.) without concern for managing system correctness.

The common shell also simplifies software by ensuring that all applications support the same API functions for data movement, reconfiguration, and health monitoring. For example, we co-designed the PCIe core and DMA engine to achieve very low latency, taking fewer than 10 μs for transfers

Figure 4. Components of the shell architecture.

of 16 KB or less, and multithreading safety. Software developers use simple *send* and *receive* calls to transmit data, and role designers simply read and write to the PCIe interface FIFOs.

The shell consumes 23% of each FPGA, although extra capacity can be obtained by discarding unused functions. If desired, partial reconfiguration allows for dynamic switching between roles while the shell remains active—even routing inter-FPGA traffic while a reconfiguration is taking place—but it comes at the cost of reduced logic area and RAM availability to the application.

3.5. Resiliency
At datacenter scales, providing resiliency in the face of hardware failures is essential given that such failures occur frequently, while demand for hardware availability is consistently high. For instance, the fabric must stay available in the presence of errors, failing hardware, reboots, and updates to the implemented algorithm. FPGAs can potentially corrupt their neighbors or crash the hosting servers if care is not taken during reconfiguration. Our reconfigurable fabric further requires a custom protocol to reconfigure groups of FPGAs, remap services to recover from failures, and report errors to the management software. In addition, ECC is used on all external memories, and SEU detection and correction is implemented on the FPGA's configuration memory.

3.6. Total cost of ownership
To balance the expected per-server performance gains versus the necessary increase in total cost of ownership, including both increased capital costs and operating expenses, we set aggressive power and cost goals to achieve a positive ROI. We are unable to give cost numbers for our production servers due to their business sensitivity; however, we can say

that adding the FPGA card and network cost less than 30% in the total cost of ownership, including a limit of 10% for total server power.

3.7. Datacenter deployment
To test this architecture on a critical production-scale datacenter service at scale, we manufactured and deployed the fabric in a production datacenter. The deployment consisted of a total of 1632 machines, that were organized into in 17 server racks. Each server uses two 12-core Intel Xeon CPUs, 64 Gbytes of DRAM, and two solid-state drives (SSDs) in addition to four hard-disk drives. The machines have a standard 10-Gbit Ethernet network card connected to a 48-port top-of-rack switch, which in turn connects to the broader datacenter network. The FPGA daughter cards and cable assemblies were tested at manufacture and again at system integration. At deployment, we discovered that seven cards (0.4%) had a hardware failure and that one of the 3264 links (0.03%) in the cable assemblies was defective. Since then, after several months of operation, we have seen no additional hardware failures.

4. APPLICATION CASE STUDY
To study the potential impact of the Catapult fabric, we ported a significant fraction of Microsoft Bing's web search ranking engine into reconfigurable hardware. Aside from being a representative datacenter-scale workload, Bing consumes a significant fraction of the datacenter capacity at Microsoft and is used to power a number of popular services such as Yahoo! Search, Apple Siri, and search on XBox One. As an interactive workload, Bing requires both low latency and high bandwidth simultaneously. Furthermore, Bing has strict resiliency requirements, is operationally complex, and is programmed using tens of thousands of lines of production-quality C++ code—making it an excellent candidate for gauging the viability of reconfigurable computing at scale in a production environment.

In our case study, we devoted the majority of our efforts to the Ranking portion of the Bing search engine, which presented the largest opportunity for hardware acceleration. Over 30K lines of C++ code were manually ported to the Catapult fabric using the Verilog HDL. The implementation generates results that are identical to software—even reproducing known software bugs. As will be discussed in further detail below, our implementation requires a total of seven FPGAs to run a single instance of the service—plus one additional spare for redundancy. Without the availability of the low-latency, high-bandwidth network that interconnects multiple FPGAs, accelerating the Ranking service would not have been feasible.

4.1. Accelerating Bing ranking using FPGAs
The Bing search engine is logically divided into multiple software services spanning a large number of servers in the datacenter. When a user's search query is processed by the Bing search engine, it is first submitted to a front-end cache service that stores and delivers the results of previously submitted and popular queries. If the search query cannot be located in the cache, it is forwarded to the Selection and

Ranking services, which perform the actual computation needed to generate search results.

The Selection service is responsible for accepting a user query and selecting which of the billions of documents (e.g., web pages) on the Internet are worthwhile candidates. The Ranking service further takes these selected documents and runs them through a sophisticated ranking algorithm that determines the order in which these documents should be presented to the user.
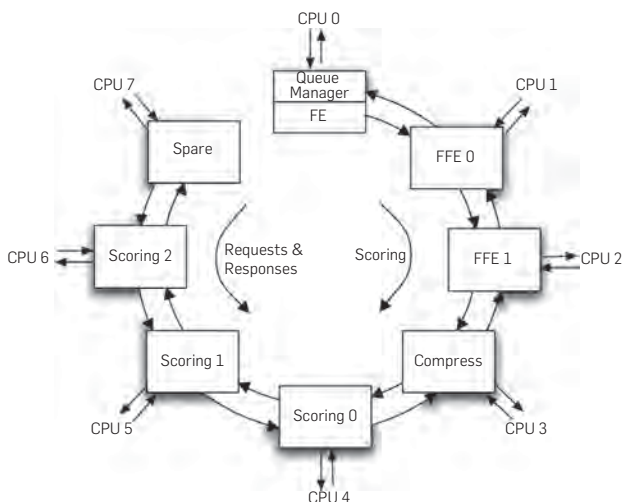
The input to the Bing Ranking algorithm is a "hit vector" that corresponds to a document-query pair arriving from the upstream Selection service. A hit vector efficiently encodes the locations in which words in a user's query appear within a given document (e.g., web page). The output of the Ranking algorithm is a document "score," which is used to determine the position in which the document is presented to the user.

Conceptually, the Bing ranking algorithm is divided into three major stages: (1) Feature Extraction (FE), (2) Free-Form Expressions (FFE), and (3) Machine-Learned Scoring (MLS). Figure 5 illustrates a hardware processing pipeline that allocates these stages to an eight-node FPGA pipeline: one FPGA for FE, two for FFEs, one for a compression stage that increases scoring engine efficiency, and three for MLS. The eighth FPGA is a spare that allows the ring to be reconfigured and rotated to keep the ranking pipeline alive in the event of a failure.

4.2. Feature extraction

In FE, interesting characteristics of a given document are dynamically extracted based on a user's search query. As a simple example, the *NumberOfOccurences* feature simply counts up the number of times a query happens to appear within a given document. In Bing Ranking, there are potentially up to thousands of features that are computed for a given document-query pair.

Our FPGA accelerator achieves a significant advantage over software using a form of multiple-instruction, single-data computation (MISD). In the FPGA, we instantiate 43 unique feature-extraction state machines that are used to compute nearly 4500 features per document-query pair in parallel. Each feature is an independent instruction stream, and the data is a single document—hence the MISD parallelism—which the FPGA can exploit far more effectively than CPUs and GPUs.
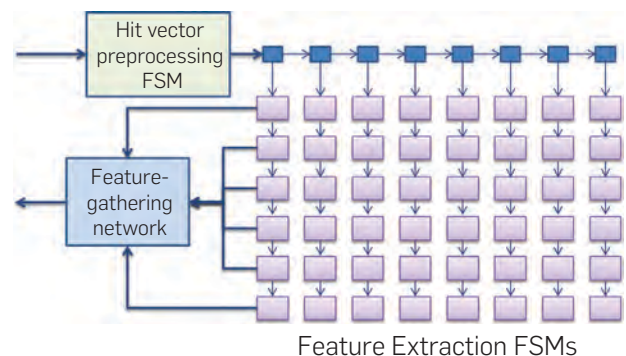
Each state machine reads each hit vector one item at a time and performs a local calculation. For some features that have similar computations, a single state machine is responsible for calculating values for multiple features. As an example, the *NumberOfOccurences* feature simply counts up how many times each term (i.e., word) in the query appears. At the end of a document, the state machine outputs all non-zero feature values; for *NumberOfOccurences*, this could be up to the number of terms in the query.

To support a large collection of state machines working in parallel on the same input data at a high clock rate, we organize the blocks into a tree-like hierarchy and replicate each input several times. Figure 6 shows the logical organization of the FE hierarchy. Hit vectors are fed into a hit vector processing state machine, which produces a series of control and data tokens that the various feature state machines process. Each state machine processes each hit vector at a rate of one to two clock cycles per token. When a state machine finishes its computation, it emits one or more feature indexes and values that are fed into the feature-gathering network that coalesces the results from the 43 state machines into a single output stream for the downstream FFE stages. Inputs to FE are double buffered to increase throughput.

4.3. Free-Form expressions

FFEs are mathematical combinations of the features extracted during the feature-extraction stage. FFEs give developers a way to create hybrid features that are not conveniently specified as feature-extraction state machines. There are typically thousands of FFEs, ranging from simple

Figure 5. Mapping of ranking roles to FPGAs on the reconfigurable fabric. Data is sent from each server to the queue manager. It is then dispatched through the seven FPGA computation stages, and the results are sent back to the source server.

Figure 6. The first stage of the ranking pipeline. Each hit vector is streamed into the hit vector preprocessing state machine, split into control and data tokens, and issued in parallel to the 43 unique feature state machines. The feature-gathering network collects generated feature and value pairs and forwards them to the next pipeline stage.

Feature Extraction FSMs

(such as adding two features) to large and complex (with thousands of operations including conditional execution and complex floating-point operators such as *ln*, *pow*, and *fpdiv*). FFEs vary greatly across models, making it impractical to synthesize customized datapaths for each expression.

One potential solution is to tile many off-the-shelf soft processor cores (such as Nios II[1]), but these single-threaded cores are inefficient at processing thousands of threads with long-latency floating-point operations in the desired amount of time per macropipeline stage (8 μs). Instead, we developed a custom multicore processor with massive multi-threading and long-latency operations in mind. The result is the FFE processor shown in Figure 7. The FFE microarchitecture is highly area efficient, letting us instantiate 60 cores on a single FPGA.

The custom FFE processor has three key characteristics that make it capable of executing all of the expressions within the required deadline. First, each core supports four simultaneous threads that arbitrate for functional units on a cycle-by-cycle basis. When one thread is stalled on a long operation such as a floating-point divide or natural log operation, other threads continue to make progress. All functional units are fully pipelined, so any unit can accept a new operation on each cycle.
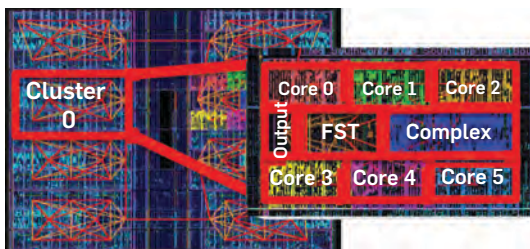
Second, rather than fair thread scheduling, threads are statically prioritized using a priority encoder. The assembler maps the expressions with the longest expected latency to thread slot 0 on all cores, then fills in slot 1 on all cores, and so forth. Once all cores have one thread in each thread slot, the remaining threads are appended to the end of previously mapped threads, starting again at thread slot 0.

Third, the longest-latency expressions are split across multiple FPGAs. An upstream FFE unit can perform part of the computation and produce an intermediate result called a metafeature. These metafeatures are sent to the downstream FFEs like any other feature, effectively replacing that part of the expression computation with a simple feature read.

4.4. Document scoring
The last stage of the pipeline is a machine-learned model evaluator that takes the features and FFEs as inputs and produces a single floating-point score. This score is sent back to the search software, and all of the resulting scores for the query are sorted and returned to the user in sorted order as the search results.

Figure 7. Free-form expressions (FFEs) placed and routed on an FPGA. Sixty cores fit on a single FPGA.

4.5. Parallelism
To overcome the slower clock frequency of FPGAs relative to CPUs and GPUs, each of the scoring stages takes advantage of two forms of parallelism that are not easily handled by the other architectures. First, each processing stage described here is configured with deep pipelines that match the amount of pipeline parallelism available in the application.

Second, FE and FFE exhibit multiple instruction single data (MISD) parallelism, a cousin of the more commonly known single instruction multiple data (SIMD) parallelism exploited by GPUs and the vector processing units in CPUs. A single source of data (the document) is operated on by a very large number of independent instruction streams (feature extractors and free form expressions for FE and FFE, respectively). SIMD architectures require the opposite—a large number of independent data elements operated on by the same instruction stream.

While SIMD architectures can efficiently process applications with MISD parallelism by batching many sets of data together, this comes at the cost of increased latency, which is often prohibitive in interactive cloud applications such as web search. As such, web ranking is an example of a cloud application that FPGAs can accelerate more effectively other parallel processing architectures.
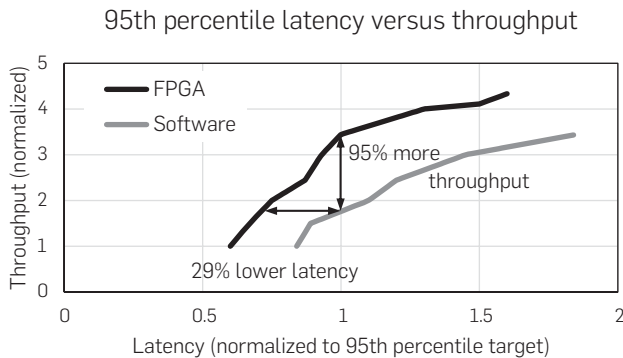
5. EVALUATION
We evaluated the Catapult fabric by deploying and measuring the described Bing ranking engine on a bed of 1632 servers with FPGAs. Six hundred and seventy-two ran the ranking service, and the other machines ran the selection service to feed documents and queries to the ranking servers. We compare the average and tail latency distributions of Bing's production-level ranker running with and without FPGAs on that bed.

User experience is dictated determined more by tail latencies rather than average latencies—users care little if their search results come back faster than expected, but they become unhappy quickly if the results are slower than expected. As such, we report performance at the latency of the 95th percentile of queries—the time at which only 5% of queries are slower. Performance results are very similar for average latency queries (50th percentile), and are even better for higher tail latencies (99th percentile and 99.9th percentile), which have the biggest impact on user experience.

Figure 8 illustrates how the FPGA-accelerated ranker substantially reduces the end-to-end scoring latency and improves throughput relative to software. There are two ways to view the performance improvements on this graph. First, for a fixed point on the *x*-axis, it shows the improvement in throughput at that specified query latency. For example, at 1.0 (which represent the maximum acceptable latency to Bing at the 95th percentile), the FPGA achieves a 95% gain in scoring throughput relative to software.

Second, for a fixed point on the *y*-axis, it shows the improvement in response time at a given throughput. At 1.0, representing the average query load on a server, the FPGA reduces query latency by 29%. The improvement in FPGA

Figure 8. Achievable performance within a given latency bound. The points on the *x*-axis at 1.0 shows the maximum sustained throughputs on both the FPGA and software while satisfying Bing's target for latency at the 95th percentile.

95th percentile latency versus throughput

scoring latency increases further at higher injection rates, because the variability of software latency increases at higher loads (due to contention in the CPU's memory hierarchy), whereas the FPGA's performance remains stable. This improved stability means that the FPGA is capable of absorbing bursts of traffic better than software alone, which may reduce the need for overprovisioning for bursty traffic.

Given that FPGAs can be used to improve both latency and throughput, Bing could reap the benefits in many ways. For example, for equivalent ranking capacity, fewer servers can be purchased. At the current average query rate, Bing could use roughly half the number of servers and still achieve their performance targets while achieving massive cost savings. As another example, the faster response time means that additional capabilities and features can be added to the software and/or hardware stack to improve the quality of searches without exceeding the maximum allowed latency. Of course, a combination of the two is also possible.

6. CONCLUSION

For many years, FPGAs have shown promise for accelerating many computational tasks. Yet despite the huge potential, they have not yet become mainstream in modern datacenters. Our goal in building the Catapult fabric was to understand what problems must be solved to operate FPGAs at datacenter scale, and whether significant performance improvements are achievable for large-scale production workloads, especially workloads that change over the lifetime of the servers.

We found that efficiently mapping a significant portion of a complex datacenter workload to FPGAs is both possible and provides a significant ROI. We showed that an at-scale deployment of FPGAs can increase ranking throughput in a production search infrastructure by 95% at comparable latency to a software-only solution, making possible both cost savings with fewer servers needed and headroom for improved search algorithms. We achieved this without breaking the homogeneous architecture of data-center servers, and without increasing the server failure rate. The added FPGA boards increased power consumption by only 10%, did

not exceed our 30% limit in an individual server's total cost of ownership, yielding a significant overall improvement in system efficiency and TCO.

Reconfigurable fabrics based on FPGAs are not the only accelerator platform that we considered. GPUs are one option for accelerating large-scale workloads, and when we first began we considered using GPUs. However, the SIMD parallelism that GPUs handle so efficiently are not a good match for latency-sensitive, but highly divergent ranking stages (such as FE). In addition, the high power consumption of GPUs meant that they couldn't be easily incorporated into conventional servers, which only have power and cooling provisioning for a standard 25W PCIe card. Instead, they are likely more appropriate for HPC environments rather than widespread datacenter deployment.

We conclude that distributed reconfigurable fabrics are a viable path forward as increases in server performance level off, and will be crucial at the end of Moore's law for continued cost and capability improvements in cloud computing. Reconfigurability is a critical means by which hardware acceleration can keep pace with the rapid rate of change in datacenter services.

Going forward, the biggest obstacle to widespread adoption of FPGAs in the datacenter is likely to be programmability. FPGA development still requires extensive hand-coding in Register Transfer Level and manual tuning. Yet we believe that incorporating careful HW/SW co-design of custom ISAs such as those used in FE and FFE, domain-specific languages such as OpenCL, FPGA-targeted C-to-gates tools, and libraries of reusable components and design patterns, will be sufficient to permit high-value services to be productively targeted to FPGAs. Longer term, improvements in FPGA architectures for computing, more integrated development tools, and the design of languages and tools that consider accelerator offload as a core functionality will be necessary to increase the programmability of these fabrics beyond teams of specialists working with large-scale service developers. Within 10–15 years, well past the end of Moore's Law, we believe that compilation to a combination of hardware and software will be commonplace. Reconfigurable systems, such as the Catapult fabric presented here, will be necessary to support these hybrid computation models.

References

1. Altera. *Nios II Processor Reference Handbook*, 13.1.0 edition, 2014.
2. Altera. *Stratix V Device Handbook*, 14.01.10 edition, 2014.
3. Baxter, R., Booth, S., Bull, M., Cawood, G., Perry, J., Parsons, M., Simpson, A., Trew, A., Mccormick, A., Smart, G., Smart, R., Cantle, A., Chamberlain, R., Genest, G. Maxwell – A 64 FPGA Supercomputer. *Eng. Lett. 16* (2008), 426–433, 2008.
4. BEECube. *BEE4 Hardware Platform*, 1.0 edition, 2011.
5. Blott, M., Vissers, K. Dataflow architectures for 10Gbps line-rate key-value stores. In *HotChips 2013* (August 2013).
6. Convey. *The Convey HC-2 Computer*, conv-12-030.2 edition, 2012.
7. Cray. *Cray XD1 Datasheet*, 1.3 edition, 2005.
8. Dennard, R., Rideout, V., Bassous, E., LeBlanc, A. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE J. Solid-State Circ.* 9, 5 (Oct. 1974), 256–268.
9. Estlick, M., Leeser, M., Theiler, J., Szymanski, J.J. Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware. In *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, FPGA'01 (New York, NY, USA, 2001). ACM.
10. Gens, F. Worldwide and Regional Public IT Cloud Services 2014–2018 Forecast (Oct. 2014).
11. George, A., Lam, H., Stitt, G. Novo-G: At the forefront of scalable reconfigurable supercomputing. *Comput. Sci. Eng. 13*, 1 (2011), 82–86.
12. Hussain, H.M., Benkrid, K., Erdogan, A.T., Seker, H. Highly parameterized K-means clustering on FPGAs: Comparative results with GPPs and GPUs. In *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs*, RECONFIG'11 (Washington, DC, USA, 2011). IEEE Computer Society.
13. IBM. *IBM PureData System for Analytics N2001*, WAD12353-USEN-01 edition, 2013.
14. Lavasani, M., Angepat, H., Chiou, D. An FPGA-based in-line accelerator for memcached. *Comput. Arch. Lett.* PP, 99 (2013), 1–1.
15. Martin, A., Jamsek, D., Agarawal, K. FPGA-based application acceleration: Case study with GZIP compression/decompression streaming engine. In *ICCAD Special Session 7C* (November 2013).
16. Microsoft. *How Microsoft Designs Its Cloud-Scale Servers*, 2014.
17. Pell, O., Mencer, O. Surviving the end of frequency scaling with reconfigurable dataflow computing. *SIGARCH Comput. Archit. News 39*, 4 (Dec. 2011).
18. Showerman, M., Enos, J., Pant, A., Kindratenko, V., Steffen, C., Pennington, R., Hwu, W. QP: A Heterogeneous Multi-accelerator Cluster. 2009.
19. SRC. *MAPstation Systems*, 70000 AH edition, 2014.
20. Yan, J., Zhao, Z.-X., Xu, N.-Y., Jin, X., Zhang, L.-T., Hsu, F.-H. Efficient query processing for web search engine with FPGAs. In *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, FCCM'12 (Washington, DC, USA, 2012). IEEE Computer Society.

Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Joo-Young Kim, Sitaram Lanka, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao and Doug Burger, Microsoft, Redmond, WA.

Derek Chiou, Microsoft and University of Texas at Austin.

Kypros Constantinides, Amazon Web Services, Boston, MA.

John Demme, Columbia University, New York, NY.

Hadi Esmaeilzadeh, Georgia Institute of Technology, Atlanta, GA.

Scott Hauck, University of Washington, Seattle.

Amir Hormati, Google, Inc., Mountain View, CA.

James Larus, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.